中南大學

CENTRAL SOUTH UNIVERSITY

操作系统应用实践

实验报告

姓　　名 _　　_

团队成员 _　　___

学　　号 _　　_

专　　业 _　　___

班　　级 _

指导教师 _　　___

编写日期 _　　___

1.需求说明

1.1. 基本需求

U 盘准备好后，请在 Windows 下直接将其格式化为 FAT16 格式。接下来，在 Linux 环境下，编写一个文件系统管理程序，用于对 U 盘上的文件进行管理。具体需求如下：

（1）实现一个目录列表函数，类似于 Linux 中的 Shell 命令 ls。函数格式为：int ud_ls()；只需包含基本命令，不必支持多个选项。

（2）设计并实现一个删除文件函数，使用要删除的文件名（在当前目录中）作为参数。函数格式为：int ud_df(char *name)；需要查找文件，遍历 FAT 中的链接，设置 FAT 中的每个簇项并将其标志为未使用，同时更新目录项。在删除文件时，需注意文件的隐藏、只读和系统属性。任何具有这些设置的文件都不能删除。

（3）设计并实现一个改变目录函数，可将当前目录切换到上一层目录或当前目录的子目录中（无需处理路径名）。函数格式为：int ud_cd(char *directory)；假设 U 盘上已存在子目录。需要在文件系统中使用一个静态变量表示当前目录，并对其进行操作。函数能够返回上一级目录，并需要使用一个静态变量在文件系统中代表当前目录的父目录。

（4）设计并实现一个创建文件函数，使用要创建的文件名和文件大小为参数。函数格式为：int ud_cf(char *filename, int size)；需要遍历 FAT 表中的链接，找出 FAT 表中的能存放下文件大小的空簇，并更新目录项。可以用 ud_ls()函数查询到所创建的文件。

1.2. 进阶需求

（1）对 ud_ls()函数进行完善，增加对全部非根目录信息的读取。在本实验中，之前只读取了一个扇区的非根目录信息，现在进行改进以支持读取全部非根目录信息。

（2）对 ud_cf()函数进行改进，使其可以向文件中写入实际内容，并根据写入的内容计算文件实际大小。

（3）增加对绝对路径和多级目录的支持。在这里，需要对输入的目录路径字符串进行解析，然后逐级查找目录。

（4）设计并实现一个删除目录函数，通常需要先判断目录是否为空目录。若目录不为空，则给出提示并删除其包含的所有子目录和文件；若是空目录则可直接删除。函数格式为：int ud_dd(char *directory)。

2. 设计说明

2.1. 结构设计

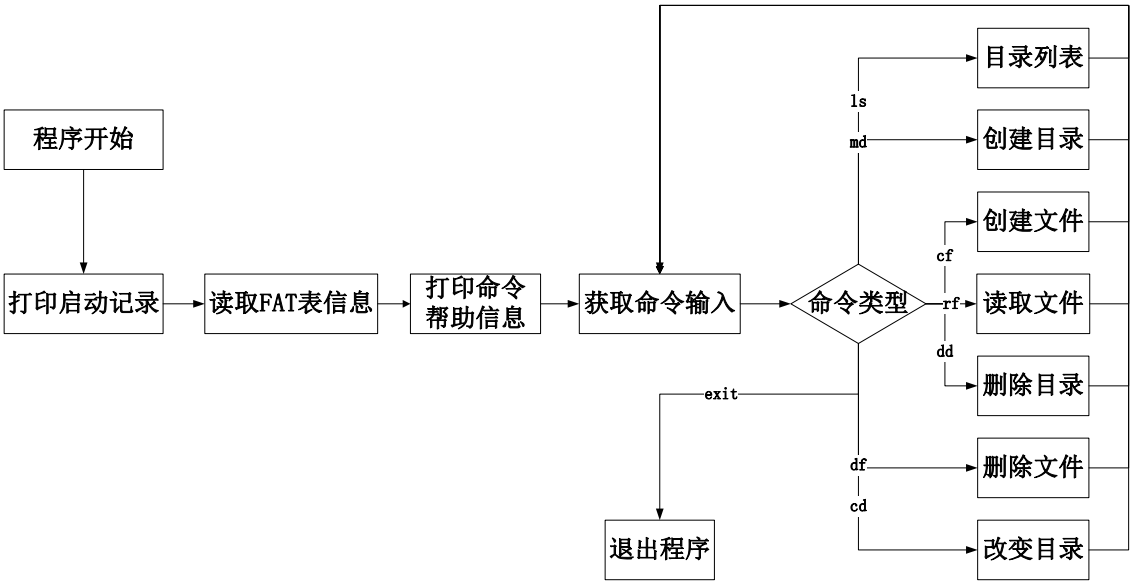

图 2.1 程序结构设计图

2.2. 功能设计

2.2.1. 重要的数据结构设计

结构名称	结构标识名称	字段信息	字段说明	结构功能描述
启动记录	BootDescriptor	unsigned char Oem_name[9]	原始设备制造商名称	以结构体形式存储 FAT 表中关于文件系统启动记录的信息
		int BytesPerSector	每扇区的字节数	
		int	每簇的扇区数	

		SectorsPerCluster		
		int ReservedSectors	保留的扇区	
		int FATs	FAT 表个数	
		int RootDirEntries	根目录个数	
		int LogicSectors	扇区个数	
		int MediaType	介质类型	
		int SectorsPerFAT	每个 FAT 表的扇区数	
		int SectorsPerTrack	每个磁道的扇区数	
		int Heads	磁头个数	
		int HiddenSectors	隐藏扇区数	
根目录项	Entry	unsigned char short_name[12]	短文件名	存储当前目录下的目录项信息
		unsigned char long_name[27]	长文件名	
		unsigned short year,month,day	目录项创建日期	
		unsigned short hour,min,sec	目录项创建时间	
		unsigned short FirstCluster	目录项首簇	
		unsigned int size	目录项大小	
		unsigned char readonly:1	读写属性	
		unsigned char hidden:1	隐藏属性	
		unsigned char system:1	系统属性	

		unsigned char vlabel:1	卷标属性	
		unsigned char subdir:1	子目录属性	
		unsigned char archive:1	归档属性	

表 2.1 结构类型设计说明表

2.2.2. 主要函数或接口设计

2.2.2.1. 接口清单

```
3.  #ifndef FILE_GLOBAL_H_
4.  #define FILE_GLOBAL_H_
5.
6.  /*************************************************************/
7.  /*全局常量定义*/
8.  #define DEVNAME "/dev/sdb8"                                  //要打开的设备文件名,
    表示当前使用的 U 盘
9.  #define DIR_ENTRY_SIZE 32                                    //目录项大小,FAT16 文
    件系统中的目录项通常是 32 字节
10. #define SECTOR_SIZE 512                                      //每个扇区大小 大多数
    存储设备以 512 字节的扇区为单位进行数据传输
11. #define CLUSTER_SIZE 512*4                                   //每个簇大小,FAT16 文
    件系统中的文件分配是以簇为单位的。设置为 4 个扇区（2KB）的簇大小是出于权衡性能和磁盘空间
    的考虑
12. #define FAT_SIZE 250*512                                     //每个 FAT 表大小
    (250ge)
13. #define FAT_ONE_OFFSET 512                                   //第一个 FAT 表起始地
    址 常见的引导扇区大小 512
14. #define FAT_TWO_OFFSET 512+250*512                           //第二个 FAT 表起始地
    址跟踪簇（clusters）分配情况的表格，其中包含了文件系统上所有簇的信息。
15. //FAT 表通常会有两个拷贝，分别称为 FAT1 和 FAT2，以提高文件系统的可靠性
16. #define ROOTDIR_OFFSET 512+250*512+250*512+512               //根目录区起始地址
17. #define DATA_OFFSET 512+250*512+250*512+512*32               //数据区起始地址
    (genmulu 32)
18.
19. /**属性位掩码**/
20. #define ATTR_READONLY 0x01
21. #define ATTR_HIDDEN 0x02
22. #define ATTR_SYSTEM 0x04
```

```
23. #define ATTR_VLABEL 0x08
24. #define ATTR_SUBDIR 0x10
25. #define ATTR_ARCHIVE 0x20
26.
27. /**时间掩码 5：6：5 **/
28. #define MASK_HOUR 0xf800
29. #define MASK_MIN 0x07e0
30. #define MASK_SEC 0x001f
31.
32. /**日期掩码**/
33. #define MASK_YEAR 0xfe00
34. #define MASK_MONTH 0x01e0
35. #define MASK_DAY 0x001f
36.
37. /*************************************************************/
38. /*全局结构体定义*/
39.
40. /**启动记录结构**/
41. typedef struct BootDescriptor
42. {
43.     unsigned char Oem_name[9];                          //0x03-0x0a
44.     int BytesPerSector;                                 //0x0b-0x0c
45.     int SectorsPerCluster;                              //0x0d
46.     int ReservedSectors;                                //0x0e-0x0f
47.     int FATs;                                           //0x10
48.     int RootDirEntries;                                 //0x11-0x12
49.     int LogicSectors;                                   //0x13-0x14
50.     int MediaType;                                      //0x15
51.     int SectorsPerFAT;                                  //0x16-0x17
52.     int SectorsPerTrack;                                //0x18-0x19
53.     int Heads;                                          //0x1a-0x1b
54.     int HiddenSectors;                                  //0x1c-0x1d
55. }BootDescriptor;
56.
57. /**根目录项结构**/
58. typedef struct Entry
59. {
60.     unsigned char short_name[12];                       //字节 0-10，11 字节的
    短文件名
61.     unsigned char long_name[27];                        //未使用，26 字节的长
    文件名
62.     unsigned short year,month,day;                      //22-23 字节
63.     unsigned short hour,minute,second;                  //24-25 字节
64.     unsigned short FirstCluster;                        //26-27 字节
```

```
65.     unsigned int size;                                  //28-31 字节
66.     /*属性位           11 字节
67.      * 7    6   5   4   3   2   1   0
68.      * N    N   A   D   V   S   H   R   N：未使用
69.      */
70.
71.     unsigned char readonly:1;                           //R(只读)
72.     unsigned char hidden:1;                             //H(隐藏)
73.     unsigned char system:1;                             //S(系统文件)
74.     unsigned char vlabel:1;                             //V(卷标)
75.     unsigned char subdir:1;                             //D(子目录)
76.     unsigned char archive:1;                            //A(存档)
77. }Entry;
78.
79. /*************************************************************/
80. /*全局变量定义*/
81.
82. int fd;                                                 //文件描述符
83. BootDescriptor bdptor;                                  //启动记录
84. Entry *curdir=NULL;                                    //当前目录
85. int dirno=0;                                           //代表目录的层数
86. Entry* fatherdir[10];                                  //父级目录
87. unsigned char fatbuf[FAT_SIZE];                        //存储 FAT 表信息
88.
89. /*************************************************************/
90. /*全局函数声明*/
91.
92. int ud_ls(int);                                        //显示所要查看目录下
    的内容
93. int ud_cd(char *);                                    //改变目录到父目录或
    子目录
94. int ud_cf(char *,int);                                //在当前目录下创建文
    件
95. int ud_md(char *);                                    //在当前目录下创建目
    录
96. int ud_df(char *);                                    //删除当前目录下的文
    件
97. int ud_dd(char *);                                    //删除当前目录下的目
    录项
98. int ud_rf(char *);                                    //读取当前目录下的文
    件内容
99.
100. void findDate(unsigned short *,unsigned short *,unsigned short *,unsigned char
     []);
```

```
101. void findTime(unsigned short *,unsigned short *,unsigned short *,unsigned char
     []);
102.
103. int readFat();                                          //读 fat 表的信息
104. int writeFat();                                         //将改变的 fat 表值写
     回 fat 表
105. void scanBootSector();                                 //浏览启动扇区
106. void scanRootEntry();                                  //浏览根目录
107. int scanEntry(char *,Entry *,int);                     //浏览当前目录
108. int getEntry(Entry *);                                 //从根目录或文件簇中
     得到文件表项
109. void formatFileName(unsigned char *);                  //文件名格式化
110. unsigned short getFatCluster(unsigned short);          //在 fat 表中获得下一
     簇的位置
111. void clearFatCluster(unsigned short);                  //清除 fat 表中的簇信
     息
112.
113. /*************************************************************/
114. #endif /* FILE_GLOBAL_H_ */
```

2.2.2.2 函数功能说明

函数名称	返回类型	参数	功能描述
ud_ls	int	int	显示当前目录的内容
ud_cd	int	char *	改变当前目录到父目录或子目录
ud_cf	int	char *	在当前目录下创建文件
		int	
ud_md	int	char *	在当前目录下创建子目录
ud_df	int	char *	删除当前目录下的文件
ud_dd	int	char *	删除当前目录下的子目录
ud_rf	int	char *	读取当前目录下的文件内容
readFat	int		读取 FAT 表信息
writeFat	int		将改变的 FAT 表值写回 FAT 表
scanBootSector	void		浏览启动扇区

scanEntry	int	char *	浏览当前目录项
		Entry *	
		int	
getEntry	int	Entry *	获取目录项在 FAT 表中的位置
getFatCluster	unsigned short	unsigned short	获取下一簇在 FAT 表中的位置
clearFatCluster	void	unsigned short	清除 FAT 表中的簇信息
formatFileName	void	unsigned char *	文件名格式化
findDate	void	unsigned short *	读取日期
		unsigned short *	
		unsigned short *	
		unsigned char []	
findTime	void	unsigned short *	读取时间
		unsigned short *	
		unsigned short *	
		unsigned char []	

表 2.2 函数功能说明表

2.2.2.3 函数调用关系

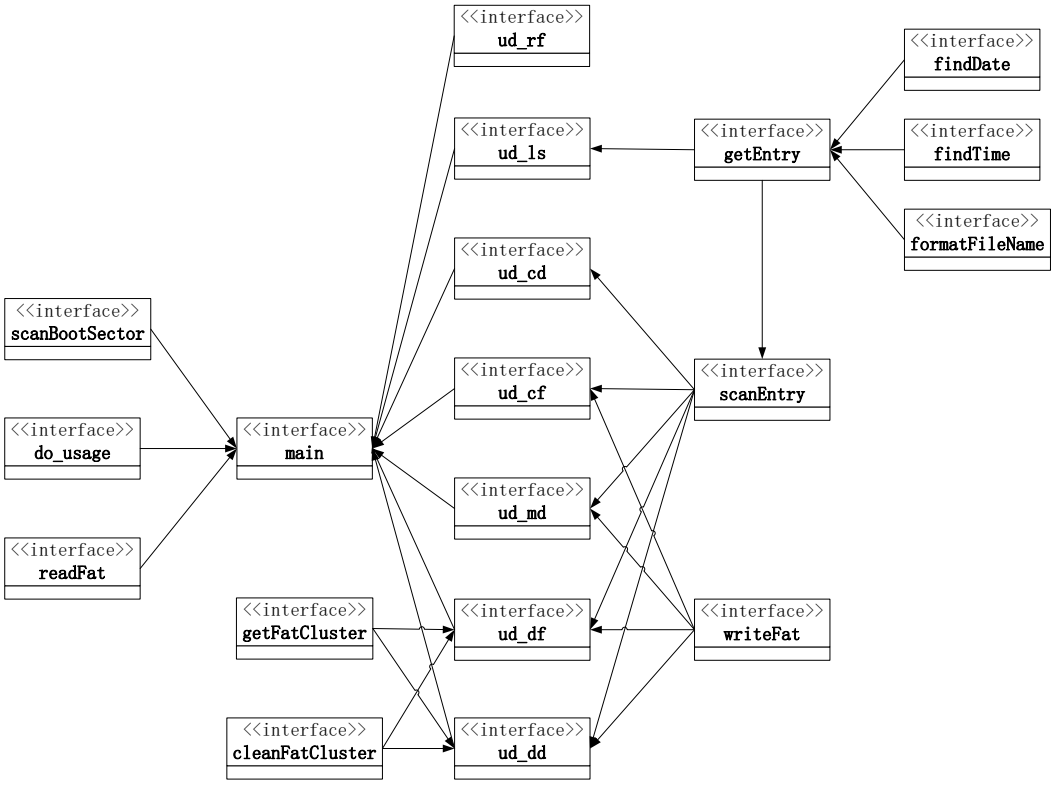

图 2.2 函数调用关系图

2.2.2.4 重要功能具体说明

2.2.2.4.1 创建目录功能的实现

本程序实现了在当前目录下创建子目录的功能。函数 ud_md()，接受参数为要在当前目录下创建的子目录的名称。创建成功则返回值为 1，失败则返回值为-1，若当前目录下存在同名子目录，则不创建目录并打印提示信息告之用户。

该函数先遍历 FAT 表，找到一个空簇作为提供给将要创建的目录结构中的首簇指向。之后获取新建的子目录在根目录区的存储地址，并将各字节信息存入。

2.2.2.4.2 删除目录功能的实现

实现了函数 ud_dd()，接受参数为要删除当前目录下的子目录名称。删除成功则返回值为 1，失败则返回值为-1。

该函数先遍历当前目录，获得要删除子目录的目录项结构信息。之后，读取目录项结构中的首簇地址字段，查看其指向地址的簇是否为文件结束簇，即判断该目录是否为空。若为空，则直接调用 clearFatCluster()函数清除 FAT 表中该

目录的记录；若不为空，则向用户发出提醒并询问用户是否删除。在删除不为空的目录时，应遍历其下的文件及子目录，递归调用相应的目录项删除函数将它们存储在 FAT 表中的信息清除掉，然后再删除最顶层的目标目录。

如是，则完成了删除一个指定目录的操作。

2.2.2.4.3 对绝对路径和多级目录的支持

在系统接收 cd 命令后，先对其后输入的转移目录路径字符串进行解析判断其属于绝对路径形式、多级目录形式还是单级目录形式。

若路径字符串以“/media/disk/”开头，则显然匹配为绝对路径。取其后的子字符串，先跳转到 U 盘文件系统下的根目录，之后按多级目录形式对取出的子字符串进行处理。

若路径字符串中含多个目录分隔号“/”，则显然匹配为多级目录。从头到尾遍历该字符串，将每两个分隔符间的目录作为子字符串取出，按单级目录形式传给 cd 函数处理。这样每处理完一个子串，则当前目录路转到该目录下，直到字符串遍历完成。

其余情况均作为单级目录处理，若字符串为“.”，则保持当前目录不变；若字符串为“..”，则跳转到当前目录所在的父一级目录，同时修改存储父级目录的记录表指向的目录地址；若字符串为目录名称，将要跳转的子目录名字符串作为参数传递给 cd 函数处理，将当前目录修改为其下的这一子目录，并将改变前的目录存入存储父级目录的记录表中。

2.2.2.4.4 创建文件函数的改进

在创建文件函数 ud_cf()中，加入了对用户写入文件内容的支持。在执行用户输入的 cf 命令时，首先获取用户输入的文件内容，通过调用 ud_cf()函数创建文件时，在 FAT 表中查找空白簇，将文件内容存入数据区相对应于找到并建立好的 FAT 表链接。

关键代码实现如下：

```
1.     printf("enter the file's data:\n");
2.             scanf("%s",data);
3.             size=strlen(data);
4.             clustersize=size/CLUSTER_SIZE;
5.
6.             if(size%CLUSTER_SIZE!=0)
7.                 clustersize++;
8.
9.             /*查询 fat 表，找到空白簇，保存在 clusterno[]中*/
```

```
            for(cluster=2;cluster<1000;cluster++)
            {
                index=cluster*2;
                if(fatbuf[index]==0x00&&fatbuf[index+1]==0x00)
                {
                    clusterno[i]=cluster;

                    i++;
                    if(i==clustersize)
                        break;
                }
            }

            /*在 fat 表中写入下一簇信息*/
            for(i=0;i<clustersize-1;i++)
            {
                index=clusterno[i]*2;
                fatbuf[index]=(clusterno[i+1]&0x00ff);
                fatbuf[index+1]=((clusterno[i+1]&0xff00)>>8);

                if(lseek(fd,(clusterno[i]-2)*CLUSTER_SIZE+DATA_OFFSET,SEEK_SET)<
0)   //随机访问文件，文件指针设置为距离文件开头 offset 个字节处
                    perror("lseek file data failed");
                if(write(fd,data+i*CLUSTER_SIZE,CLUSTER_SIZE)<0)
                    perror("write failed");
            }
```

文件大小是由用户输入文本数据后根据其大小求得。在 FAT 表进行写入下一簇信息的同时，在数据区相应簇中写入文件数据。这样 FAT 表在建立对数据区数据存储的索引时，也完成了文件数据的存入。

2.2.2.4.3 读取文件函数的实现

实现了函数 ud_rf()用于读取磁盘中的文件内容。在获取了所要读取的文件的目录项信息后，根据首簇地址在 FAT 表中顺序查找数据区该文件内容的存储地址，读出并打印。

关键代码实现如下：

```
    seed=pentry->FirstCluster;   // 获取文件的第一个簇号
        size=pentry->size;           // 获取文件大小
        page=size/CLUSTER_SIZE;     // 计算文件页数
        buf=(unsigned char *)malloc(sizeof(unsigned char)*size);

```

```
        // 逐页读取文件内容
        for(i=0;i<page;i++)
        {
            if((ret=lseek(fd,(seed-2)*CLUSTER_SIZE+DATA_OFFSET,SEEK_SET))<0)
                perror("lseek cluster_addr failed");
            read(fd,buf+i*CLUSTER_SIZE,CLUSTER_SIZE);
            seed=getFatCluster(seed);   // 获取下一个簇号
        }

        // 读取最后一页文件内容
        if((ret=lseek(fd,(seed-2)*CLUSTER_SIZE+DATA_OFFSET,SEEK_SET))<0)
            perror("lseek cluster_addr failed");
        read(fd,buf+i*CLUSTER_SIZE,size-i*CLUSTER_SIZE);

        // 输出文件内容到标准输出
        for(i=0;i<size;i++)
            printf("%c",buf[i]);
        printf("\n");
```

按目录表项中文件大小字段值先求得文件数据所占数据区的簇数后，按 FAT 表中索引依次从所在的各簇中读取数据，转存打印。

2.2.2.4.4 目录列表函数的完善

检查用户输入的 ls 命令后所带参数项，若为-all，则传递参数 1 给函数 ud_ls()，打印所有非根目录的信息；若为-cur，则传递参数 0 给函数 ud_ls()，打印当前目录的信息（这与 dir 命令功能相同）。

非根目录列表：在函数 ud_ls()中通过不断调用函数 ud_cd()来改变当前目录，并打印即时当前目录的信息，即实现了所有非根目录列表显示的功能。

当前目录列表：在函数中从当前目录起始地址开始遍历，获取当前目录下的目录项，并打印显示，即实现了当前目录列表显示的功能。

3. 测试和使用说明

3.1 使用说明

3.1.1 开发环境

Linux 操作系统版本：Ubuntu 9.04

集成开发环境：eclipse-SDK-3.4.2-linux-gtk

GCC 编译器：gcc-4.3.3

3.1.2 运行环境

Linux 操作系统平台

FAT16 格式的 U 盘

3.1.3 安装说明

将程序文件夹中的可执行文件 filesystem.out 复制到 Linux 系统的桌面上，插入一张在 windows 系统下格式化为 FAT16 格式的 U 盘，在终端中修改当前路径进入桌面，执行命令./filesystem.out，即可运行文件管理主程序，它读取 FAT 表中启动记录信息并显示后，打印命令提示符‘>’，等待用户输入文件管理命令并进行相关处理。

3.2 测试说明

3.2.1 测试用例

3.2.1.1 输入描述

插入格式化为 FAT16 格式的 U 盘，运行程序的可执行文件。

看到启动记录被正常打印后，用户可根据随之打印出的帮助信息中文件管理相关命令格式，输入相应命令，执行相应的文件管理操作。

附：文件管理格式命令说明

命令名称	命令格式	描述说明
目录列表命令	ls -a 或 ls -all	列表显示所有非根目录项。
目录列表命令	ls -c 或 ls -cur	列表显示当前目录下所有目录项
改变目录命令	cd <dir>	将当前目录改变为参数字符串所代表的目录路径。
创建文件命令	cf <filename> <size>	在当前目录下新建一个文件，命令后面第一个参数为文件名，第二个参数为预设定的大小。执行该命令后，用户可随即输入文件内容并保存。

创建目录命令	md <direcotory>	在当前目录下新建一个目录，参数代表目录名称。
删除文件命令	df <filename>	删除当前目录下的文件，参数字符串代表要删除的文件名称。
删除目录命令	dd <directory>	删除当前目录下的目录，参数字符串代表要删除的目录名称。若该目录为空则直接删除，若不为空，则询问用户，得到删除的确认后，该其下子目录项全部删除。
读取文件命令	rf <filename>	打开当前目录下的一个文件，并将文件中的内容打印出来，参数字符串代表要查看内数的文件名称。

表 3.1 文件管理格式命令说明表

3.2.1.2 输出描述

程序启动后，首先打印了磁盘的启动记录。其后，依照用户输入的文件管理命令，执行相应操作，打印与其相符的信息，确保文件管理被正确执行。

3.2.1.3 运行结果

```
Oem_name        MSDOS5.0
BytesPerSector          512
SectorsPerCluster       32
ReservedSector          4
FATs                    2
RootDirEntries          512
LogicSectors            0
MediaType               248
SectorPerFAT            246
SectorPerTrack          63
Heads                   255
HiddenSectors           63
please input a command, including followings:
     ls <-a|-all>               list all non-root entries
     ls <-c|-cur>               list all entries under current directory
     cd <dir>                   change direcotry
     cf <filename> <size>       create a file
     md <directory>             make a directory
     df <file>                  delete a file
     dd <directory>             delete a directory
     rf <filename>              read a file
     exit                       exit this system
>
```

图 3.1 磁盘启动记录 运行截图

```
文件(F)  编辑(E)  查看(V)  终端(T)  帮助(H)
FATs                    191
RootDirEntries          1563
LogicSectors            22352
MediaType               185
SectorPerFAT            485
SectorPerTrack          42227
Heads                   48843
HiddenSectors           1982
please input a command, including followings:
        ls <-a|-all>            list all non-root entries
        ls <-c|-cur>            list all entries under current directory
        cd <dir>                change direcotry
        cf <filename> <size>    create a file
        md <directory>          make a directory
        df <file>               delete a file
        dd <directory>          delete a directory
        rf <filename>           read a file
        exit                    exit this system
>md aa
>ls -c
Root_dir
        name            date            time            cluster size    attr
              AA        2009-6-17       16:30:36        2       0       dir
>
```

图 3.2 目录列表命令记录 运行截图

```
文件(F)  编辑(E)  查看(V)  终端(T)  帮助(H)
        df <file>               delete a file
        dd <directory>          delete a directory
        rf <filename>           read a file
        exit                    exit this system
>ls -c
Root_dir
        name            date            time            cluster size    attr
              AA        2009-6-17       16:30:36        2       0       dir
              TE        2009-6-17       16:34:36        4       0       dir
>cd /media/disk/aa
>md uu
>ls -c
Root_dir
        name            date            time            cluster size    attr
              AA        2009-6-17       16:30:36        2       0       dir
              TE        2009-6-17       16:34:36        4       0       dir
              UU        2009-6-17       16:36:2         5       0       dir
>cd /media/disk/aa/
>md uu
>ls -c
AA_dir
        name            date            time            cluster size    attr
              UU        2009-6-17       16:36:28        6       0       dir
>
```

图 3.3 切换目录命令 运行截图

```
文件(F)  编辑(E)  查看(V)  终端(T)  帮助(H)
Root_dir
        name            date            time            cluster size    attr
              AA        2009-6-17       16:30:36        2       0       dir
              OO        2009-6-17       16:31:8         3       4       file
>cf test 32
enter the file's data:
hello world..
>please input a command, including followings:
        ls <-a|-all>            list all non-root entries
        ls <-c|-cur>            list all entries under current directory
        cd <dir>                change direcotry
        cf <filename> <size>    create a file
        md <directory>          make a directory
        df <file>               delete a file
        dd <directory>          delete a directory
        rf <filename>           read a file
        exit                    exit this system
>ls -c
Root_dir
        name            date            time            cluster size    attr
              AA        2009-6-17       16:30:36        2       0       dir
              OO        2009-6-17       16:31:8         3       4       file
            TEST        2009-6-17       16:31:58        4       5       file
>
```

图 3.4 创建文件命令 运行截图

4. 核心 C 程序

```
1.  /*
2.   * file_system.c
3.   */
4.
5.  #include <stdio.h>
6.  #include <unistd.h>
7.  #include <stdlib.h>
8.  #include <sys/types.h>
9.  #include <sys/stat.h>
10. #include <fcntl.h>
11. #include <string.h>
12. #include <time.h>
13. #include <ctype.h>
14. #include "file_global.h"
15. //字节翻转（大小端）
16. #define RevByte(low,high) ((high)<<8|(low))
17. //字的翻转（四个字节反一下）
18. #define RevWord(lowest,lower,higher,highest) ((highest)<<24|(higher)<<16|(lo
    wer)<<8|lowest)
19.
20. /*
```

```
21.          * 读 fat 表的信息，存入 fatbuf[]中
22.          */
23.         //lseek 是一个用于在文件中定位文件指针位置的系统调用。lseek 用于移动文件描述
    符 fd 关联的文件指针到指定的位置。
24.         // 具体来说，lseek 的原型是：
25.         // off_t lseek(int fd, off_t offset, int whence);
26.         //SEEK_SET 就是初始位置
27.         int readFat()
28.         {
29.             if(lseek(fd,FAT_ONE_OFFSET,SEEK_SET)<0)
30.             {
31.                 perror("lseek failed");
32.                 return -1;
33.             }
34.             //read 函数从当前文件指针位置读取 FAT_SIZE 字节的数据到 fatbuf 缓冲区中
35.             if(read(fd,fatbuf,FAT_SIZE)<0)
36.             {
37.                 perror("read failed");
38.                 return -1;
39.             }
40.
41.             return 1;
42.         }
43.
44.         /*
45.          * 将改变的 fat 表值写回 fat 表
46.          */
47.         int writeFat()
48.         {
49.             if(lseek(fd,FAT_ONE_OFFSET,SEEK_SET)<0)
50.             {
51.                 perror("lseek failed");
52.                 return -1;
53.             }
54.             if(write(fd,fatbuf,FAT_SIZE)<0)
55.             {
56.                 perror("read failed");
57.                 return -1;
58.             }
59.             if(lseek(fd,FAT_TWO_OFFSET,SEEK_SET)<0)
60.             {
61.                 perror("lseek failed");
62.                 return -1;
63.             }
```

```
64.             if((write(fd,fatbuf,FAT_SIZE))<0)
65.             {
66.                 perror("read failed");
67.                 return -1;
68.             }
69.             return 1;
70.         }
71.
72.         /*打印启动项记录*/
73.         //扫描引导扇区（Boot Sector）
74.         void scanBootSector()
75.         {
76.             unsigned char buf[SECTOR_SIZE];
77.             int ret,i;
78.
79.             if((ret=read(fd,buf,SECTOR_SIZE))<0)
80.                 perror("read boot sector failed");
81.             for(i=0;i<8;i++)
82.                 bdptor.Oem_name[i]=buf[i+0x03];
83.             bdptor.Oem_name[i]='\0';
84.
85.             // 这部分代码通过从 buf 数组中提取相应的字节，
86.             // 将引导扇区的信息存储到结构体 bdptor 中。例如，
87.             // bdptor.Oem_name 是一个字符数组，用于存储引导扇
88.             // 区的 OEM 名称，通过将 buf 中的相应字节复制到 Oem_name 中来实现。
89.             bdptor.BytesPerSector=RevByte(buf[0x0b],buf[0x0c]);
90.             bdptor.SectorsPerCluster=buf[0x0d];
91.             bdptor.ReservedSectors=RevByte(buf[0x0e],buf[0x0f]);
92.             bdptor.FATs=buf[0x10];
93.             bdptor.RootDirEntries=RevByte(buf[0x11],buf[0x12]);
94.             bdptor.LogicSectors=RevByte(buf[0x13],buf[0x14]);
95.             bdptor.MediaType=buf[0x15];
96.             bdptor.SectorsPerFAT=RevByte( buf[0x16],buf[0x17]);
97.             bdptor.SectorsPerTrack=RevByte(buf[0x18],buf[0x19]);
98.             bdptor.Heads=RevByte(buf[0x1a],buf[0x1b]);
99.             bdptor.HiddenSectors=RevByte(buf[0x1c],buf[0x1d]);
100.
101.            printf("Oem_name \t\t%s\n"
102.                "BytesPerSector \t\t%d\n"
103.                "SectorsPerCluster \t%d\n"
104.                "ReservedSector \t\t%d\n"
105.                "FATs \t\t\t%d\n"
106.                "RootDirEntries \t\t%d\n"
107.                "LogicSectors \t\t%d\n"
```

```
108.                "MediaType \t\t%d\n"
109.                "SectorPerFAT \t\t%d\n"
110.                "SectorPerTrack \t\t%d\n"
111.                "Heads \t\t\t%d\n"
112.                "HiddenSectors \t\t%d\n",
113.                bdptor.Oem_name,
114.                bdptor.BytesPerSector,
115.                bdptor.SectorsPerCluster,
116.                bdptor.ReservedSectors,
117.                bdptor.FATs,
118.                bdptor.RootDirEntries,
119.                bdptor.LogicSectors,
120.                bdptor.MediaType,
121.                bdptor.SectorsPerFAT,
122.                bdptor.SectorsPerTrack,
123.                bdptor.Heads,
124.                bdptor.HiddenSectors);
125.        }
126.
127.        /*
128.         * 参数：entryname 类型：char
129.         *       pentry      类型：Entry *
130.         *       mode        类型：int，mode=1，为目录表项；mode=0，为文件
131.         * 返回值：偏移值大于 0，则成功；-1，则失败
132.         * 功能：搜索当前目录，查找文件或目录项
133.         */
134.        int scanEntry(char *entryname,Entry *pentry,int mode)
135.        {
136.            int ret,offset,i;
137.            int cluster_addr;
138.            char uppername[80];
139.
140.            for(i=0;i<strlen(entryname);i++)
141.                uppername[i]=toupper(entryname[i]);
142.            uppername[i]='\0';
143.
144.            if(curdir==NULL)        //扫描根目录
145.            {
146.                if((ret=lseek(fd,ROOTDIR_OFFSET,SEEK_SET))<0)
147.                    perror("lseek ROOTDIR_OFFSET failed");
148.                offset=ROOTDIR_OFFSET;
149.
150.                while(offset<DATA_OFFSET)
151.                {
```

```
152.                    ret=getEntry(pentry);
153.                    offset+=abs(ret);
154.
155.                    if(pentry->subdir==mode&&strcmp((char *)pentry->short_name,upper
    name)==0)
156.                        return offset;
157.                }
158.            }
159.            else                    //扫描子目录
160.            {
161.                cluster_addr=DATA_OFFSET+(curdir->FirstCluster-2)*CLUSTER_SIZE;
162.                if((ret=lseek(fd,cluster_addr,SEEK_SET))<0)
163.                    perror("lseek cluster_addr failed");
164.                offset=cluster_addr;
165.
166.                while(offset<cluster_addr+CLUSTER_SIZE)
167.                {
168.                    ret=getEntry(pentry);
169.                    offset+=abs(ret);
170.
171.                    if(pentry->subdir==mode&&strcmp((char *)pentry->short_name,upper
    name)==0)
172.                        return offset;
173.                }
174.            }
175.
176.            return -1;
177.        }
178.
179.        /*
180.         * 参数：entry，类型：Entry *
181.         * 返回值：成功，则返回偏移值；失败：返回负值
182.         * 功能：从根目录或文件簇中得到文件表项
183.         */
184.        int getEntry(Entry *pentry)
185.        {
186.            int ret,i;
187.            int count=0; // 记录读取的字节数
188.            unsigned char buf[DIR_ENTRY_SIZE],info[2];
189.
190.            /*读一个目录表项，即 32 字节*/
191.            if((ret=read(fd,buf,DIR_ENTRY_SIZE))<0)  // 读取失败时输出错误信息
192.                perror("read entry failed");
193.            count+=ret;  // 记录读取的字节数
```

```
194.
195.            if(buf[0]==0xe5||buf[0]==0x00)
196.                return -1*count;  // 如果目录项为空或已删除，则返回负值
197.            else
198.            {
199.                /* buf 读的是文件名，在 FAT 文件系统中，如果目录项的第一个字节是 0x0f，则表示这是一个长文件名的条目。循环读取并丢弃这些长文件名的条目。长文件名，忽略掉 */
200.                /*长文件名，忽略掉*/
201.                while(buf[11]==0x0f)
202.                {
203.                    if((ret=read(fd,buf,DIR_ENTRY_SIZE))<0)  //read 是系统调用，fd 是文件描述符
204.                        perror("read root dir failed");
205.                    count+=ret; // 记录读取的字节数
206.                }
207.
208.                /*命名格式化，注意结尾的'\0'*/
209.                for (i=0;i<=10;i++)
210.                    pentry->short_name[i]=buf[i];
211.                pentry->short_name[i]='\0';  // 确保字符串结尾有'\0'
212.
213.                formatFileName(pentry->short_name);  // 格式化文件名
214.
215.                info[0]=buf[22];
216.                info[1]=buf[23];
217.                findTime(&(pentry->hour),&(pentry->minute),&(pentry->second),info);
218.
219.                info[0]=buf[24];
220.                info[1]=buf[25];
221.                findDate(&(pentry->year),&(pentry->month),&(pentry->day),info);
222.
223.                // 根据文件属性位设置 Entry 结构中的标志
224.                pentry->FirstCluster=RevByte(buf[26],buf[27]);
225.                pentry->size=RevWord(buf[28],buf[29],buf[30],buf[31]);
226.
227.                pentry->readonly=(buf[11]&ATTR_READONLY)?1:0;
228.                pentry->hidden=(buf[11]&ATTR_HIDDEN)?1:0;
229.                pentry->system=(buf[11]&ATTR_SYSTEM)?1:0;
230.                pentry->vlabel=(buf[11]&ATTR_VLABEL)?1:0;
231.                pentry->subdir=(buf[11]&ATTR_SUBDIR)?1:0;
232.                pentry->archive=(buf[11]&ATTR_ARCHIVE)?1:0;
233.
234.                return count;  // 返回读取的字节数
```

```
235.            }
236.        }
237.
238.        /*日期*/
239.        void findDate(unsigned short *year,unsigned short *month,unsigned short *day
    ,unsigned char info[2])
240.        {
241.            int date;
242.            date=RevByte(info[0],info[1]);
243.
244.            *year=((date&MASK_YEAR)>>9)+1980;
245.            *month=((date&MASK_MONTH)>>5);
246.            *day=(date&MASK_DAY);
247.        }
248.
249.        /*时间*/
250.        //Hour（小时）：  从高位开始占用 5 位，即位 11 到 15。
251.        // Minute（分钟）：  中间的 6 位，即位 5 到 10。
252.        // Second（秒）：  低位占用 5 位，即位 0 到 4。
253.        //用掩码剔除不需要的部分，保留需要的，再移位
254.        //秒字段通常以 2 秒的粒度进行存储，所以需要乘以 2 来获取真实的秒数
255.        void findTime(unsigned short *hour,unsigned short *min,unsigned short *sec,u
    nsigned char info[2])
256.        {
257.            int time;
258.            time=RevByte(info[0],info[1]);
259.
260.            *hour=((time&MASK_HOUR)>>11);
261.            *min=(time&MASK_MIN)>>5;
262.            *sec=(time&MASK_SEC)*2;
263.        }
264.
265.        /*文件名格式化，便于比较*/
266.        void formatFileName(unsigned char *name)
267.        {
268.            unsigned char *p=name;
269.            while(*p!='\0')
270.                p++;
271.            p--;
272.            while(*p==' ')
273.                p--;
274.            p++;
275.            *p='\0';
276.        }
```

```
277.
278.        /*
279.         * 参数：prev，类型：unsigned char
280.         * 返回值：下一簇
281.         * 功能：在 fat 表中获得下一簇的位置
282.         */
283.        unsigned short getFatCluster(unsigned short prev)
284.        {
285.            unsigned short next;
286.            int index;
287.
288.            index=prev*2;
289.            next=RevByte(fatbuf[index],fatbuf[index+1]);
290.
291.            return next;
292.        }
293.
294.        /*
295.         * 参数：cluster，类型：unsigned short
296.         * 返回值：void
297.         * 功能：清除 fat 表中的簇信息
298.         */
299.        void clearFatCluster(unsigned short cluster)
300.        {
301.            int index;
302.            index=cluster*2;
303.
304.            fatbuf[index]=0x00;
305.            fatbuf[index+1]=0x00;
306.        }
307.
308.        /*
309.         * 参数：mode，类型：int
310.         * 返回值：1，成功；-1，失败
311.         * 功能：显示所要查看目录下的内容。
312.         * 说明：mode 值为 0 时，查看当前目录信息；mode 值为 1 时，查看所有的非根目录信息
313.         */
314.        int ud_ls(int mode)
315.        {
316.            int ret,offset,cluster_addr;
317.            int i;
318.            Entry entry;
319.            unsigned char buf[DIR_ENTRY_SIZE];
320.
```

```
321.            if((ret=read(fd,buf,DIR_ENTRY_SIZE))<0)
322.                perror("read entry failed");
323.
324.            if(mode==1)
325.            {
326.                printf("All_Non-Root_dir\n");
327.                printf("\tname\t\tdate\t\ttime\t\tcluster\tsize\tattr\n");
328.
329.                for(i=0;i<100;i++)
330.                {//遍历把读到的东西全打出来
331.                    cluster_addr=DATA_OFFSET+i*CLUSTER_SIZE; //一簇一簇的读
332.                    if((ret=lseek(fd,cluster_addr,SEEK_SET))<0)
333.                        perror("lseek cluster_addr failed");
334.
335.                    offset=cluster_addr;
336.
337.                    /*只读一簇的内容*/
338.                    while(offset<cluster_addr+CLUSTER_SIZE)
339.                    {
340.                        ret=getEntry(&entry);    //读取文件表项存到 entry，并返回偏移量表示已经读取，这一簇的信息已经获取准备给下一簇用
341.                        offset+=abs(ret);
342.
343.                        if(ret>0)
344.                        {
345.                            printf("%16s\t"
346.                                "%d-%d-%d\t"
347.                                "%d:%d:%d  \t"
348.                                "%d\t"
349.                                "%d\t"
350.                                "%s\n",
351.                                entry.short_name,
352.                                entry.year,entry.month,entry.day,
353.                                entry.hour,entry.minute,entry.second,
354.                                entry.FirstCluster,
355.                                entry.size,
356.                                (entry.subdir)?"dir":"file");
357.                        }
358.                    }
359.                }
360.                return 1;
361.            }
362.
363.            //模式 1 结束 ， 下面是模式 0
```

```
364.            if(curdir==NULL)
365.                printf("Root_dir\n");//根目录
366.            else
367.                printf("%s_dir\n",curdir->short_name);//对应的名字
368.
369.            printf("\tname\t\tdate\t\ttime\t\tcluster\tsize\tattr\n");
370.
371.            if(curdir==NULL)    //显示根目录区
372.            {
373.                /*将 fd 定位到根目录区的起始地址*/
374.                if((ret=lseek(fd,ROOTDIR_OFFSET,SEEK_SET))<0)
375.                    perror("lseek ROOTDIR_OFFSET failed");
376.
377.                offset=ROOTDIR_OFFSET;
378.
379.                /*从根目录区开始遍历，直到数据区起始地址*/
380.                while(offset<DATA_OFFSET)    //读完
381.                {
382.                    ret=getEntry(&entry);//成功读取
383.                    offset+=abs(ret);
384.
385.                    if(ret>0)
386.                    {
387.                        printf("%16s\t"
388.                            "%d-%d-%d\t"
389.                            "%d:%d:%d  \t"
390.                            "%d\t"
391.                            "%d\t"
392.                            "%s\n",
393.                            entry.short_name,
394.                            entry.year,entry.month,entry.day,
395.                            entry.hour,entry.minute,entry.second,
396.                            entry.FirstCluster,
397.                            entry.size,
398.                            (entry.subdir)?"dir":"file");
399.                    }
400.                }
401.            }
402.            else                //示当前目录（且非 root）
403.            {
404.                cluster_addr=DATA_OFFSET+(curdir->FirstCluster-2)*CLUSTER_SIZE;  //
    簇的编号通常从 2 开始（FAT 文件系统采用簇链表的方式的方式
405.                if((ret=lseek(fd,cluster_addr,SEEK_SET))<0)
406.                    perror("lseek cluster_addr failed");
```

```
407.
408.                offset=cluster_addr;
409.
410.                /*只读一簇的内容*/
411.                while(offset<cluster_addr+CLUSTER_SIZE)
412.                {
413.                    ret=getEntry(&entry);
414.                    offset+=abs(ret);
415.
416.                    if(ret>0)
417.                    {
418.                        printf("%16s\t"
419.                            "%d-%d-%d\t"
420.                            "%d:%d:%d  \t"
421.                            "%d\t"
422.                            "%d\t"
423.                            "%s\n",
424.                            entry.short_name,
425.                            entry.year,entry.month,entry.day,
426.                            entry.hour,entry.minute,entry.second,
427.                            entry.FirstCluster,
428.                            entry.size,
429.                            (entry.subdir)?"dir":"file");
430.                    }
431.                }
432.            }
433.
434.            return 0;
435.        }
436.
437.        /*
438.         * 参数：dir,类型：char
439.         * 返回值：1，成功；-1，失败
440.         * 功能：改变目录到父目录或子目录
441.         */
442.        int ud_cd(char *directory)
443.        {
444.            Entry *pentry;  // 用于存储扫描到的目录项信息
445.            int ret;        // 用于存储函数调用返回值
446.
447.            // 如果目录名为当前目录（"."），则无需改变目录，直接返回成功
448.            if(strcmp(directory,".")==0)
449.            {
450.                return 1;
```

```
451.            }
452.
453.            // 如果目录名为上一级目录（".."），且当前目录为空（根目录），则无需改变目录，直接返回成功
454.            if(strcmp(directory,"..")==0&&curdir==NULL)
455.                return 1;
456.
457.             // 如果目录名为上一级目录（".."）且当前目录不为空，返回上一级目录
458.            if(strcmp(directory,"..")==0&&curdir!=NULL)
459.            {
460.                curdir=fatherdir[dirno];
461.                dirno--;
462.                return 1;
463.            }
464.            //否则就是进入当前的一个文件夹
465.            // 分配内存以存储目录项信息
466.            pentry=(Entry *)malloc(sizeof(Entry));
467.
468.            // 扫描目录，获取目录项信息
469.            ret=scanEntry(directory,pentry,1);
470.            if(ret<0)
471.            {
472.                // 扫描失败，输出提示信息，释放内存，返回失败
473.                printf("no such directory\n");
474.                free(pentry);
475.                return -1;
476.            }
477.
478.            // 目录号加一，保存当前目录到父目录数组，更新当前目录为扫描到的目录项
479.            dirno++;
480.            fatherdir[dirno]=curdir;
481.            curdir=pentry;
482.            return 1;   // 返回成功
483.        }
484.
485.        /*
486.         * 参数：filename       类型：char *，创建文件的名称
487.         *       size          类型：int，文件的大小
488.         * 返回值：1，成功；-1，失败
489.         * 功能：在当前目录下创建文件
490.         */
491.        int ud_cf(char *filename,int size)
492.        {
493.            Entry *pentry;
```

```
494.            int ret,i=0,cluster_addr,offset;
495.            unsigned short cluster,clusterno[FAT_SIZE];
496.            unsigned char c[DIR_ENTRY_SIZE];
497.            int index,clustersize;
498.            unsigned char buf[DIR_ENTRY_SIZE];
499.            struct tm *local;
500.            time_t t;
501.            int cutime,cudate;
502.            unsigned char data[size];
503.
504.            t=time(NULL);
505.            local=localtime(&t);
506.            //Hour（小时）：  从高位开始占用 5 位，即位 11 到 15。
507.            // Minute（分钟）：  中间的 6 位，即位 5 到 10。
508.            // Second（秒）：  低位占用 5 位，即位 0 到 4。
509.            //用掩码剔除不需要的部分，保留需要的，再移位
510.            //秒字段通常以 2 秒的粒度进行存储，所以需要乘以 2 来获取真实的秒数
511.            //获得文件系统的时间戳
512.            cutime=local->tm_hour*2048+local->tm_min*32+local->tm_sec/2;
513.            cudate=(local->tm_year+1900-1980)*512+(local->tm_mon+1)*32+local->tm_mda
    y;
514.
515.            pentry=(Entry *)malloc(sizeof(Entry));
516.            ret=scanEntry(filename,pentry,0);           //扫描根目录，是否已存在该文件
    名
517.            if(ret<0)    //不存在该文件名
518.            {
519.                printf("enter the file's data:\n");
520.                scanf("%s",data);
521.                size=strlen(data);
522.                clustersize=size/CLUSTER_SIZE;
523.
524.                if(size%CLUSTER_SIZE!=0)
525.                    clustersize++;
526.
527.                /*查询 fat 表，找到空白簇，保存在 clusterno[]中*/
528.                for(cluster=2;cluster<1000;cluster++)
529.                {
530.                    index=cluster*2;
531.                    if(fatbuf[index]==0x00&&fatbuf[index+1]==0x00)
532.                    {
533.                        clusterno[i]=cluster;
534.
535.                        i++;
```

```
536.                         if(i==clustersize)
537.                             break;
538.                     }
539.                 }
540.
541.                 /*在 fat 表中写入下一簇信息*/
542.                 for(i=0;i<clustersize-1;i++)
543.                 {
544.                     index=clusterno[i]*2;
545.                     fatbuf[index]=(clusterno[i+1]&0x00ff);
546.                     fatbuf[index+1]=((clusterno[i+1]&0xff00)>>8);
547.
548.                     if(lseek(fd,(clusterno[i]-2)*CLUSTER_SIZE+DATA_OFFSET,SEEK_SET)<
    0)   //随机访问文件，文件指针设置为距离文件开头 offset 个字节处
549.                         perror("lseek file data failed");
550.                     if(write(fd,data+i*CLUSTER_SIZE,CLUSTER_SIZE)<0)
551.                         perror("write failed");
552.                 }
553.                 /*最后一簇写入 0xffff*/
554.                 index=clusterno[i]*2;
555.                 fatbuf[index]=0xff;
556.                 fatbuf[index+1]=0xff;
557.
558.                 if(lseek(fd,(clusterno[i]-2)*CLUSTER_SIZE+DATA_OFFSET,SEEK_SET)<0)
559.                     perror("lseek file data failed");
560.                 if(write(fd,data+i*CLUSTER_SIZE,size-i*CLUSTER_SIZE)<0)
561.                     perror("write failed");
562.
563.                 if(curdir==NULL)        //往根目录下写文件
564.                 {
565.                     if((ret=lseek(fd,ROOTDIR_OFFSET,SEEK_SET))<0)
566.                         perror("lseek ROOTDIR_OFFSET failed");
567.                     offset=ROOTDIR_OFFSET;
568.                     while(offset<DATA_OFFSET)    //目录区和数据区分开，防止越界
569.                     {
570.                         if((ret=read(fd,buf,DIR_ENTRY_SIZE))<0)
571.                             perror("read entry failed");
572.
573.                         offset+=abs(ret);
574.
575.                         if(buf[0]!=0xe5&&buf[0]!=0x00)  //0x00 表示目录项为空，0xe5 表示
    目录项曾被使用，现已删除  不能用要跳过，更新 offset
576.                         {
577.                             while(buf[11]==0x0f)
```

```
578.                            {
579.                                if((ret=read(fd,buf,DIR_ENTRY_SIZE))<0)
580.                                    perror("read root dir failed");
581.                                offset+=abs(ret);
582.                            }
583.                        }
584.                        else            //找出空目录项或已删除的目录项
585.                        {
586.                            offset-=abs(ret);   //回退一个目录项
587.
588.                            /**编写创建的目录项**/
589.                            for(i=0;i<=strlen(filename);i++)
590.                                c[i]=toupper(filename[i]);  //小写转大写
591.                            for(;i<=10;i++)
592.                                c[i]=' ';       //补' ': 格式
593.
594.                            c[0x0b]=0x01;
595.
596.                            /*写时间和日期*/
597.                            c[0x16]=(cutime&0x00ff);
598.                            c[0x17]=((cutime&0xff00)>>8);
599.                            c[0x18]=(cudate&0x00ff);
600.                            c[0x19]=((cudate&0xff00)>>8);
601.
602.                            /*写第一簇的值*/
603.                            c[0x1a]=(clusterno[0]&0x00ff);
604.                            c[0x1b]=((clusterno[0]&0xff00)>>8);
605.
606.                            /*写文件的大小*/
607.                            c[0x1c]=(size&0x000000ff);
608.                            c[0x1d]=((size&0x0000ff00)>>8);
609.                            c[0x1e]=((size&0x00ff0000)>>16);
610.                            c[0x1f]=((size&0xff000000)>>24);
611.
612.                            if(lseek(fd,offset,SEEK_SET)<0)
613.                                perror("lseek ud_cf failed");
614.                            if(write(fd,&c,DIR_ENTRY_SIZE)<0)
615.                                perror("write failed");
616.
617.                            free(pentry);
618.                            if(writeFat()<0)
619.                                exit(1);
620.
621.                            return 1;
```

```
622.                         }
623.                     }
624.                 }
625.                 else    //当前目录不为空，在当前目录下新建文件
626.                 {
627.                     cluster_addr=(curdir->FirstCluster-2)*CLUSTER_SIZE+DATA_OFFSET;

628.                     if((ret=lseek(fd,cluster_addr,SEEK_SET))<0)
629.                         perror("lseek cluster_addr failed");
630.                     offset=cluster_addr;
631.                     while(offset<cluster_addr+CLUSTER_SIZE)  //控制不超过簇的大小限
    制
632.                     {
633.                         if((ret=read(fd,buf,DIR_ENTRY_SIZE))<0)
634.                             perror("read entry failed");
635.
636.                         offset+=abs(ret);
637.
638.                         if(buf[0]!=0xe5&&buf[0]!=0x00)
639.                         {
640.                             while(buf[11]==0x0f)
641.                             {
642.                                 if((ret=read(fd,buf,DIR_ENTRY_SIZE))<0)
643.                                     perror("read root dir failed");
644.                                 offset+=abs(ret);
645.                             }
646.                         }
647.                         else
648.                         {
649.                             offset-=abs(ret);
650.                             for(i=0;i<=strlen(filename);i++)
651.                                 c[i]=toupper(filename[i]);
652.                             for(;i<=10;i++)
653.                                 c[i]=' ';
654.
655.                             c[0x0b]=0x01;
656.
657.                             /*写时间和日期*/
658.                             c[0x16]=(cutime&0x00ff);
659.                             c[0x17]=((cutime&0xff00)>>8);
660.                             c[0x18]=(cudate&0x00ff);
661.                             c[0x19]=((cudate&0xff00)>>8);
662.
663.                             /*写第一簇的值*/
```

```
664.                            c[0x1a]=(clusterno[0]&0x00ff);
665.                            c[0x1b]=((clusterno[0]&0xff00)>>8);
666.
667.                            /*写文件的大小*/
668.                            c[0x1c]=(size&0x000000ff);
669.                            c[0x1d]=((size&0x0000ff00)>>8);
670.                            c[0x1e]=((size&0x00ff0000)>>16);
671.                            c[0x1f]=((size&0xff000000)>>24);
672.
673.                            if(lseek(fd,offset,SEEK_SET)<0)
674.                                perror("lseek ud_cf failed");
675.                            if(write(fd,&c,DIR_ENTRY_SIZE)<0)
676.                                perror("write failed");
677.
678.                            free(pentry);
679.                            if(writeFat()<0)
680.                                exit(1);
681.
682.                            return 1;
683.                        }
684.                    }
685.                }
686.            }
687.            else
688.            {
689.                printf("This filename is exist\n");
690.                free(pentry);
691.                return -1;
692.            }
693.
694.            return 1;
695.        }
696.        /*
697.         * 参数：filename       类型：char *，待查找文件名称
698.         * 返回值：1，成功；-1，失败
699.         * 功能：读取当前目录下文件的内容
700.         */
701.        int ud_rf(char *filename)
702.        {
703.            Entry *pentry;  // 用于存储扫描到的文件项信息
704.            int ret,size,page;  // 存储函数调用返回值、文件大小和文件页数
705.            int i=0;            // 循环变量
706.            unsigned char *buf;     // 用于存储文件内容的缓冲区
707.            unsigned short seed;    // 存储文件的第一个簇号
```

```
708.
709.            // 分配内存以存储文件项信息
710.            pentry=(Entry *)malloc(sizeof(Entry));
711.
712.            //扫描当前目录查找文件
713.            ret=scanEntry(filename,pentry,0);
714.            if(ret<0)
715.            {
716.                // 扫描失败，输出提示信息，释放内存，返回失败
717.                printf("no such file\n");
718.                free(pentry);
719.                return -1;
720.            }
721.
722.            seed=pentry->FirstCluster;    // 获取文件的第一个簇号
723.            size=pentry->size;            // 获取文件大小
724.            page=size/CLUSTER_SIZE;      // 计算文件页数
725.            buf=(unsigned char *)malloc(sizeof(unsigned char)*size);
726.
727.            // 逐页读取文件内容
728.            for(i=0;i<page;i++)
729.            {
730.                if((ret=lseek(fd,(seed-2)*CLUSTER_SIZE+DATA_OFFSET,SEEK_SET))<0)
731.                    perror("lseek cluster_addr failed");
732.                read(fd,buf+i*CLUSTER_SIZE,CLUSTER_SIZE);
733.                seed=getFatCluster(seed);    // 获取下一个簇号
734.            }
735.
736.            // 读取最后一页文件内容
737.            if((ret=lseek(fd,(seed-2)*CLUSTER_SIZE+DATA_OFFSET,SEEK_SET))<0)
738.                perror("lseek cluster_addr failed");
739.            read(fd,buf+i*CLUSTER_SIZE,size-i*CLUSTER_SIZE);
740.
741.            // 输出文件内容到标准输出
742.            for(i=0;i<size;i++)
743.                printf("%c",buf[i]);
744.            printf("\n");
745.
746.            return 0;    // 返回成功
747.        }
748.
749.        /*
750.         * 参数：directory，类型：char *
751.         * 返回值：1，成功；－1，失败
```

```
752.         * 功能：在当前目录下创建目录
753.         */
754.        int ud_md(char *directory)
755.        {
756.            Entry *pentry;
757.            int ret,i=0,cluster_addr,offset;
758.            unsigned short cluster;
759.            unsigned char c[DIR_ENTRY_SIZE];
760.            int index;
761.            unsigned char buf[DIR_ENTRY_SIZE];
762.            struct tm *local;
763.            time_t t;
764.            int cutime,cudate;
765.
766.            t=time(NULL);
767.            local=localtime(&t);
768.            cutime=local->tm_hour*2048+local->tm_min*32+local->tm_sec/2;
769.            cudate=(local->tm_year+1900-1980)*512+(local->tm_mon+1)*32+local->tm_mda
    y;
770.
771.            pentry=(Entry *)malloc(sizeof(Entry));
772.            ret=scanEntry(directory,pentry,1);          //扫描根目录，是否已存在该目录
    名
773.
774.            if(ret<0)
775.            {
776.                /*查询 fat 表，找到空白簇，保存在 clusterno[]中*/
777.                for(cluster=2;cluster<1000;cluster++)
778.                {
779.                    index=cluster*2;
780.                    if(fatbuf[index]==0x00&&fatbuf[index+1]==0x00)
781.                    {
782.                        /*在 fat 表中写入 0xffff*/
783.                        fatbuf[index]=0xff;
784.                        fatbuf[index+1]=0xff;
785.
786.                        break;
787.                    }
788.                }
789.
790.                if(curdir==NULL)        //往根目录下写目录
791.                {
792.                    if((ret=lseek(fd,ROOTDIR_OFFSET,SEEK_SET))<0)
793.                        perror("lseek ROOTDIR_OFFSET failed");
```

```
794.                    offset=ROOTDIR_OFFSET;
795.                    while(offset<DATA_OFFSET)
796.                    {
797.                        if((ret=read(fd,buf,DIR_ENTRY_SIZE))<0)
798.                            perror("read entry failed");
799.
800.                        offset+=abs(ret);
801.
802.                        if(buf[0]!=0xe5&&buf[0]!=0x00)
803.                        {
804.                            while(buf[11]==0x0f)
805.                            {
806.                                if((ret=read(fd,buf,DIR_ENTRY_SIZE))<0)
807.                                    perror("read root dir failed");
808.                                offset+=abs(ret);
809.                            }
810.                        }
811.                        else            //找出空目录项或已删除的目录项
812.                        {
813.                            offset-=abs(ret);
814.                            for(i=0;i<=strlen(directory);i++)
815.                                c[i]=toupper(directory[i]);
816.                            for(;i<=10;i++)
817.                                c[i]=' ';
818.
819.                            c[0x0b]=0x10;
820.
821.                            /*写时间和日期*/
822.                            c[0x16]=(cutime&0x00ff);
823.                            c[0x17]=((cutime&0xff00)>>8);
824.                            c[0x18]=(cudate&0x00ff);
825.                            c[0x19]=((cudate&0xff00)>>8);
826.
827.                            /*写第一簇的值*/
828.                            c[0x1a]=(cluster&0x00ff);
829.                            c[0x1b]=((cluster&0xff00)>>8);
830.
831.                            /*写目录的大小*/
832.                            c[0x1c]=(0x00000000&0x000000ff);
833.                            c[0x1d]=((0x00000000&0x0000ff00)>>8);
834.                            c[0x1e]=((0x00000000&0x00ff0000)>>16);
835.                            c[0x1f]=((0x00000000&0xff000000)>>24);
836.
837.                            if(lseek(fd,offset,SEEK_SET)<0)
```

```
838.                                perror("lseek ud_md failed");
839.                            if(write(fd,&c,DIR_ENTRY_SIZE)<0)
840.                                perror("write failed");
841.
842.                            free(pentry);
843.                            if(writeFat()<0)
844.                                exit(1);
845.
846.                            return 1;
847.                        }
848.                    }
849.                }
850.                else
851.                {
852.                    cluster_addr=(curdir->FirstCluster-2)*CLUSTER_SIZE+DATA_OFFSET;

853.                    if((ret=lseek(fd,cluster_addr,SEEK_SET))<0)
854.                        perror("lseek cluster_addr failed");
855.                    offset=cluster_addr;
856.                    while(offset<cluster_addr+CLUSTER_SIZE)
857.                    {
858.                        if((ret=read(fd,buf,DIR_ENTRY_SIZE))<0)
859.                            perror("read entry failed");
860.
861.                        offset+=abs(ret);
862.
863.                        if(buf[0]!=0xe5&&buf[0]!=0x00)
864.                        {
865.                            while(buf[11]==0x0f)
866.                            {
867.                                if((ret=read(fd,buf,DIR_ENTRY_SIZE))<0)
868.                                    perror("read root dir failed");
869.                                offset+=abs(ret);
870.                            }
871.                        }
872.                        else
873.                        {
874.                            offset=offset-abs(ret);
875.                            for(i=0;i<=strlen(directory);i++)
876.                                c[i]=toupper(directory[i]);
877.                            for(;i<=10;i++)
878.                                c[i]=' ';
879.
880.                            c[0x0b]=0x10;
```

```
881.
882.                            /*写时间和日期*/
883.                            c[0x16]=(cutime&0x00ff);
884.                            c[0x17]=((cutime&0xff00)>>8);
885.                            c[0x18]=(cudate&0x00ff);
886.                            c[0x19]=((cudate&0xff00)>>8);
887.
888.                            /*写第一簇的值*/
889.                            c[0x1a]=(cluster&0x00ff);
890.                            c[0x1b]=((cluster&0xff00)>>8);
891.
892.                            /*写目录的大小*/
893.                            c[0x1c]=(0x00000000&0x000000ff);
894.                            c[0x1d]=((0x00000000&0x0000ff00)>>8);
895.                            c[0x1e]=((0x00000000&0x00ff0000)>>16);
896.                            c[0x1f]=((0x00000000&0xff000000)>>24);
897.
898.                            if(lseek(fd,offset,SEEK_SET)<0)
899.                                perror("lseek ud_md failed");
900.                            if(write(fd,&c,DIR_ENTRY_SIZE)<0)
901.                                perror("write failed");
902.
903.                            free(pentry);
904.                            if(writeFat()<0)
905.                                exit(1);
906.
907.                            return 1;
908.                        }
909.                    }
910.                }
911.            }
912.            else
913.            {
914.                printf("This directory is exist\n");
915.                free(pentry);
916.                return -1;
917.            }
918.
919.            return 1;
920.        }
921.
922.        /*
923.         * 参数：filename，类型：char *
924.         * 返回值：1，成功；-1，失败
```

```
925.         * 功能：删除当前目录下的文件
926.         */
927.        int ud_df(char *filename)
928.        {
929.            Entry *pentry;
930.            int ret;
931.            unsigned char c;
932.            unsigned short seed,next;
933.
934.            pentry=(Entry *)malloc(sizeof(Entry));
935.            ret=scanEntry(filename,pentry,0);            //扫描当前目录查找文件
936.
937.            if(ret<0)
938.            {
939.                printf("no such file\n");
940.                free(pentry);
941.                return -1;
942.            }
943.
944.            /*清除 fat 表项*/
945.            seed=pentry->FirstCluster;    //获取文件的第一个簇号(文件数据在 FAT 中的起始位
    置)
946.            while((next=getFatCluster(seed))!=0xffff)    //遍历文件的所有簇,直到 FAT 表的
    结束标记
947.            {
948.                clearFatCluster(seed);
949.                seed=next;
950.            }
951.
952.            clearFatCluster(seed);
953.
954.            /*清除目录表项*/
955.            c=0xe5; //标记目录项曾被使用，现已删除
956.
957.            if(lseek(fd,ret-0x20,SEEK_SET)<0)
958.                perror("lseek ud_df failed");
959.            if(write(fd,&c,1)<0)
960.                perror("write failed");
961.
962.            free(pentry);
963.            if(writeFat()<0)
964.                exit(1);
965.            return 1;
966.        }
```

```
967.
968.        /*
969.         * 参数：directory，类型：char *
970.         * 返回值：1，成功；-1，失败
971.         * 功能：删除当前目录下指定的目录
972.         */
973.        int ud_dd(char *directory)
974.        {
975.            Entry *pentry;
976.            int ret;
977.            unsigned char c;
978.            char ch;
979.            unsigned short seed,next;
980.
981.            pentry=(Entry *)malloc(sizeof(Entry));
982.            ret=scanEntry(directory,pentry,1);          //扫描当前目录查找目录
983.
984.            if(ret<0)
985.            {
986.                printf("no such directory\n");
987.                free(pentry);
988.                return -1;
989.            }
990.
991.            /*清除 fat 表项*/
992.            seed=pentry->FirstCluster;
993.            //if(getFatCluster(seed)!=0xffff)
994.            //{
995.                //printf("This directory is not empty, do you want to delete it? (Y/
    N) ");
996.
997.                //ch=getchar();
998.            //  if(ch=='Y'||ch=='y')
999.                //  ;
1000.               //else if(ch=='N'||ch=='n')
1001.                   //return -1;
1002.               //while(ch!='\n')
1003.                   //ch=getchar();
1004.           //}
1005.
1006.           while((next=getFatCluster(seed))!=0xffff)
1007.           {
1008.               clearFatCluster(seed);
1009.               seed=next;
```

```
1010.           }
1011.
1012.           clearFatCluster(seed);
1013.
1014.           /*清除目录表项*/
1015.           c=0xe5; //标记目录项曾被使用，现已删除
1016.
1017.           if(lseek(fd,ret-0x20,SEEK_SET)<0)
1018.               perror("lseek ud_dd failed");
1019.           if(write(fd,&c,1)<0)
1020.               perror("write failed");
1021.
1022.                   if(lseek(fd,ret-0x40,SEEK_SET)<0)
1023.               perror("lseek ud_dd failed");
1024.           if(write(fd,&c,1)<0)
1025.               perror("write failed");
1026.
1027.           free(pentry);
1028.           if(writeFat()<0)
1029.               exit(1);
1030.           return 1;
1031.       }
1032.
1033.       /*使用方法*/
1034.       void do_usage()
1035.       {
1036.           printf("please input a command, including followings:\n");
1037.           printf("\tls <-a|-all>\t\tlist all non-root entries\n"
1038.                   "\tls <-c|-cur>\t\tlist all entries under current directory\n"
1039.                   "\tcd <dir>\t\tchange direcotry\n"
1040.                   "\tcf <filename> <size>\tcreate a file\n"
1041.                   "\tmd <directory>\t\tmake a directory\n"
1042.                   "\tdf <file>\t\tdelete a file\n"
1043.                   "\tdd <directory>\t\tdelete a directory\n"
1044.                   "\trf <filename>\t\tread a file\n"
1045.                   "\texit\t\t\texit this system\n");
1046.       }
1047.
1048.       int main()
1049.       {
1050.           char input[10];
1051.           int size=0;
1052.           int i,j,k,num;
1053.           char name[30],subname[30];
```

```
1054.
1055.           // 打开指定的设备文件以进行读写操作，
1056.           //open 函数是一个系统调用，用于打开文件或者创建文件。DEVNAME 是文件名，O_RDWR 是打开文件的模式，表示以可读写的方式打开文件。
1057.           if((fd=open(DEVNAME,O_RDWR))<0)
1058.               perror("open failed");
1059.
1060.           // 调用，读取第一个 512 字节（保存扇区存储的整体存储信息）
1061.           scanBootSector();
1062.
1063.           // 读取文件分配表（FAT）
1064.           if(readFat()<0)
1065.               exit(1);
1066.
1067.           // 显示用法信息（打印）
1068.           do_usage();
1069.
1070.           // 主命令处理循环
1071.           while(1)
1072.           {
1073.               memset(name,0,30);  // name 数组的前 30 个字节都设置为 0。（显示效果）
1074.               printf(">");
1075.               scanf("%s",input);
1076.
1077.               // 检查用户输入的各种命令
1078.               if(strcmp(input,"exit")==0)
1079.                   break;
1080.               else if(strcmp(input,"\n")==0)
1081.                   continue;
1082.               else if(strcmp(input,"ls")==0)
1083.               {
1084.                   scanf("%s",name);
1085.                   // 判断用户选择查看的类型
1086.                   if(strcmp(name,"-a")==0||strcmp(name,"-all")==0) //查看所以的非根目录信息
1087.                       ud_ls(1);
1088.                   else
1089.                       ud_ls(0);
1090.               }
1091.               else if(strcmp(input,"dir")==0) //查看当前目录信息
1092.                   ud_ls(0);
1093.               else if(strcmp(input,"cd")==0)
1094.               {
1095.                   scanf("%s",name);
```

```
1096.                   // 根据输入路径更改目录
1097.                   if(strncmp(name,"/media/disk/",12)==0)  // 处理
    以 "/media/disk/" 开头的绝对路径
1098.                   {
1099.                       if(name[strlen(name)-1] != '/')
1100.                           strcat(name, "/");
1101.                       curdir=NULL;
1102.                       dirno=0;
1103.                       for(i=12,num=12;i<30&&name[i]!='\0';i++)
1104.                       {
1105.                           if(name[i]=='/')
1106.                           {   // 提取子目录名称
1107.                               memset(subname,0,30);
1108.                               for(j=num,k=0;j<i;j++,k++)
1109.                                   subname[k]=name[j];
1110.                               if(ud_cd(subname)<0) //逐层进入目标目录
1111.                                   break;
1112.                               num=i+1;
1113.                           }
1114.                       }
1115.                   }
1116.                   else if(strncmp(name,"./",2)==0)    // 处理以 "./" 开头的相对路
    径  处理儿子
1117.                   {
1118.                       if(name[strlen(name)-1] != '/')
1119.                           strcat(name, "/");
1120.                       ud_cd(".");
1121.                       for(i=2,num=2;i<30&&name[i]!='\0';i++)
1122.                       {
1123.                           if(name[i]=='/')
1124.                           {   // 提取子目录名称
1125.                               memset(subname,0,30);
1126.                               for(j=num,k=0;j<i;j++,k++)
1127.                                   subname[k]=name[j];
1128.                               if(ud_cd(subname)<0)
1129.                                   break;
1130.                               num=i+1;
1131.                           }
1132.                       }
1133.                   }
1134.                   // 处理以 "../" 开头的相对路径
1135.                   else if(strncmp(name,"../",3)==0)   //相对路径，父级目录
1136.                   {
1137.                       if(name[strlen(name)-1] != '/')
```

```
1138.                           strcat(name, "/");
1139.                       ud_cd("..");
1140.                       for(i=3,num=3;i<30&&name[i]!='\0';i++)
1141.                       {
1142.                           if(name[i]=='/')
1143.                           {   // 提取子目录名称
1144.                               memset(subname,0,30);
1145.                               for(j=num,k=0;j<i;j++,k++)
1146.                                   subname[k]=name[j];
1147.                               if(ud_cd(subname)<0)
1148.                                   break;
1149.                               num=i+1;
1150.                           }
1151.                       }
1152.                   }
1153.                   // 处理其他情况（当前目录的子目录）
1154.                   else
1155.                       ud_cd(name);
1156.               }
1157.               else if(strcmp(input,"cf")==0)
1158.               {
1159.                   scanf("%s",name);
1160.                   scanf("%s",input);
1161.                   size=atoi(input);   //字符串转 Int
1162.                   // 使用指定的名称和大小创建新文件
1163.                   ud_cf(name,size);
1164.               }
1165.               else if(strcmp(input,"md")==0)
1166.               {
1167.                   scanf("%s",name);
1168.                   // 使用指定的名称创建新目录
1169.                   ud_md(name);
1170.               }
1171.               else if(strcmp(input,"df")==0)
1172.               {
1173.                   scanf("%s",name);
1174.                   // 删除指定文件
1175.                   ud_df(name);
1176.               }
1177.               else if(strcmp(input,"dd")==0)
1178.               {
1179.                   scanf("%s",name);
1180.                   // 删除指定的目录
1181.                   ud_dd(name);
```

```
1182.             }
1183.             else if(strcmp(input,"rf")==0)
1184.             {
1185.                 scanf("%s",name);
1186.                 // 读取指定文件内容
1187.                 ud_rf(name);
1188.             }
1189.             else
1190.                 // 对于无法识别的命令，显示用法信息
1191.                 do_usage();
1192.         }
1193.
1194.         return 0;
1195.     }
```